Simulating Vivid 3D Solid Textures from 2D Growable Patterns

Yisong Chen and Horace H S Ip
Image Computing Group, Department of Computer Science, City University of Hong Kong

Abstract

An efficient model-independent 3D texture synthesis algorithm based on texture growing and texture turbulence is presented to simulate vivid 3D solid textures from 2D growable texture patterns. Given a 2D texture pattern of some growable material, our algorithm is able to create a tileable anisotropic 3D texture pattern to simulate the natural property of the material. Target objects are directly dipped into the 3D texture pattern to generate creative, sculpture like models that can be presented with reasonable interactive frame rates. Additionally, our method is conceptually simple, computationally fast, and storage efficient. To the best of our knowledge, this is the first approach that transfers a given 2D texture naturally to point rendering systems.

Keywords: texture synthesis, fractal, point rendering.

1 Introduction

Texture transfer over arbitrary surfaces or volumes has attracted much interest within the past few years. In this paper we propose a method to simulate vivid solid texture from 2D growable texture patterns. We use the concept of texture turbulence and texture growing to simulate the growth of specified 2D texture and procedurally produce tileable 3D solid texture pattern. With this in hand, model-independent 3D texture mapping becomes very easy because all one needs to do is to dip the object into the 3D texture pattern. This method is particularly suitable for simulating natural solid texture patterns such as marbles and woods to create vivid virtual artworks like woodcarvings and stone carvings. Since our technique is able to operate at frame rate, it is applicable to interactive applications, like real-time scrulpting in VR applications. Moreover, it offers one good solution to the problem of texture synthesis on point rendering systems [Grossman1998], for which most polygon surface based texturing methods are not suitable.

The remainder of the paper is organized as follows. Related work is briefly introduced in Section 2. Section 3 details our texture-synthesis method. Simulating results are given in section 4. Finally, Section 5 concludes the paper and discusses future work.

2 Related work

One important branch of texture synthesis is to decorate complex 3D scenes through solid texturing. Perlin and Peachey independently invented the concept of solid texture [Perlin1985, Peachy1985]. Solid texturing is suitable for simulating surface textures that arise from their internal structure and is ideal for rendering objects that have been carved out of a block of material such as wood or granite. The key problem of solid texturing is how to generate 3D texture pattern. Procedural texturing is perhaps the most popular method for achieving this goal [Ebert1998][Turk1991]. It can produce very impressive solid textures with a nicely designed procedural process. Procedural methods are more or less related to stochastic technique, in which white noise plays a very important role [Cabral1993][Wijk1991]. However, it is difficult to simulate some existing texture patterns with procedural methods. Although some attempts have been made to handle this problem [Peachy1985][Lewis1989], most of them are too simple to simulate complex textures naturally. Another drawback of solid texturing is that it suffers from heavy computational and storage burden when working on a large texture space. Octree texture techniques to some extent alleviate this burden [Bensen2002][DeBry2002]. However, it is achieved at the expense of model-independence and rendering efficiency.

In this paper, we present a stochastic 3D procedural approach to transfer 2D growable texture pattern to tileable 3D texture pattern. The algorithm uses the idea of texture growing and texture turbulence to generate translated, warped 2D texture patterns from the original pattern and then concatenates them to form a continuous, tileable 3D texture cube. The major advantage of our algorithm is that it is model-independent and can give rise to impressive rendering results. It is well suited for rendering of complex models of even very large amounts of rendering primitives.

3 Simulating 3D texture pattern by texture growing and texture turbulence

Growable texture is one important class of textures that can grow in 3D space to simulate many real world materials. The simplest idea to extend a given 2D growable texture patterns is to concatenate the original texture image in the z-direction that adds a third dimension to form the 3D texture [Peachy1985]. However, simple extrusion lacks the ability of simulating complex scenes naturally, and artifacts can become remarkably perceptible when the method works on most natural growable textures such as woods and marbles.

Our objective is to overcome this drawback by simulating the real-life growth of natural materials in a procedural manner. Specifically, our approach is designed such that during the process of creating the 3D texture cube, the growing of the texture in the z-direction should go along a more random path that can simulate the actual growth of the solid material better. Additionally, small warps between neighboring frames are expected to pursue better realism. That is, the texture pattern of the frame currently in consideration should not be a strict clone, but a perturbed version of the previous one. These two ideas lead to two essential concepts of our work, texture growing and texture turbulence. For convenience's sake, we will address the problem of texture turbulence first in section 3.1 and then return to the problem of texture growing in section 3.2.

3.1 Texture turbulence

The technique of image perturbation [Ebert1998] can be used to implement texture warping. The first attempt is to yield each frame by giving the previous frame an appropriate perturbation. If we consider the z-dimension as the time dimension and each new generated image as a frame, then the problem may be converted to temporal or dynamic texture synthesis [Soatto2001]. For convenience we will treat the variable z and t equivalently. The spatio-temporal autoregressive (STAR) model that is widely used in temporal texture and flow visualization is helpful for creating the visual sense of motion texture [Wang2002]. Let I(x,y,t) be the intensity of a pixel (x,y) at time t, the corresponding STAR model has the form

$$I(x,y,t) = \sum_{i=1}^{p} a_i I(x+\delta x_i, y+\delta y_i, t+\delta t_i) + a(x,y,t) \quad (1)$$

where the signal I(x,y,t) is modeled as a linear combination of lagged values of itself plus a Gaussian white noise process a(x,y,t). Such a model can be considered as an extension from a causal Gaussian Markov random model used in texture modeling by adding the time dimension.

Unfortunately, although this approach performs well in simulating several natural scenes such as fire and water, it is not well suited for generating 3D texture cube of most rigid objects. First, the existence of the noise term in the formulation limits the STAR model's usage to simulating only fluid like scenes. As shown by Wijk [Wijk2002], with the proceeding of the time dimension, the effect of the original texture image becomes weaker and weaker, while the effect of the noise term becomes greater and greater. Second, turbulence itself has the effect of blurring the input image [Cabral1993]. Continued turbulence of previous neighboring frame will unavoidably ruin more and more the original texture structure as the texture grows in the z-direction. Accordingly, generating a new texture frame by disturbing a couple of previous frames is not a good choice in our application.

To avoid the disadvantage of accumulated blurring, our solution is to generate every texture frame directly from the original texture frame by giving it an appropriate turbulence that maintains the continuity between neighboring frames. Although the turbulence will slightly blur the texture pattern, it will not cause serious accumulated distortion as in the STAR model because each frame is just a turbulence of the original frame and the effects of blurring will not propagate. The turbulence, therefore, keeps the major structural feature of the original texture pattern while changing its appearance in a random mannar to simulate the features of a lot of natural textures. Our texture computing equation has the following form:

$$I(x,y,z_i) = I(x+dx_i, y+dy_i, z_0) \quad (2)$$

where dx_i and dy_i are vectors that indicate the direction and strength of the turbulence of the i-th frame in the z-direction. The parameter z_0 means that every frame is directly warped from the very first frame, i.e. the given texture image. The noise term in equation (1) is removed to avoid the impact of the noise.

At this point the only requirement is to choose a turbulence function that enforces the continuity between neighboring frames. Fortunately, the well-known turbulence function of Perlin fits the requirement quite well as long as the parameters are carefully selected. In most of our experiments, we adopt the following equations to calculate the turbulence of a given pixel:

$$\begin{aligned} &dx = mscale*turbulence((float)x,(float)y,(float)z,tfactor) \\ &dy = mscale*turbulence((float)z,(float)y,(float)x,tfactor) \end{aligned} \quad (3)$$

where (dx,dy) is the displacement vector of the current pixel from its original position, (x,y,z) is the position vector of the current pixel in the 3D cube, and mscale and tfactor are two parameters controlling the property of the turbulence. turbulence() is the well-known turbulence function introduced by Perlin [Perlin1985]. Equation (3) can be easily discretized in the z-direction to calculate dx_i and dy_i of equation (2).

Note that the use of the reordered coordinate values of the current pixel as the parameters of the turbulence function is the key idea of our texture warping algorithm. This successfully guarantees that the resulting 3D texture cube has a random, yet continuous look throughout. We call this method texture turbulence.

3.2 Texture growing

Although texture turbulence imparts a certain degree of creativity to the original texture pattern, the turbulence is only constrained to a local region because the center position of the turbulence remains unchanged on each frame along the z-direction. This is not consistent with the growing process of many natural materials. Although there are several procedural approaches to simulate evolution of natural scenes such as particle system and L-system, they suffer from similar drawbacks of the STAR model and can hardly meet the requirement here. To simulate such a growing process, we define a unique texture path each time we create a 3D texture cube. Namely, for each pixel of the given texture pattern, one path is generated in the z-direction to denote the path along which it travels from its start point on the front surface of the 3D cube to its end point on the back surface. The path is the same for all pixels to maintain the consistency of the global texture motion. Texture path should be a 2-dimensional vector function defined in z axes that satisfies the following equation:

$$V(z_n) = V(z_0) + TexturePath(z_n) \quad (4)$$

where V(Zi) is a 2D position vector that denotes the pixel position on the i-th frame of the 3D texture cube in the z-direction, and TexturePath() is a 2D motion vector that describes the motion of the pixel through all frames in the 3D cube.

The function TexturePath() can be defined with different methods, thus providing the freedom to simulate different growing styles. In forming a natural simulation, it should be continuous, yet also have some property of randomness. We accomplished two different TexturePath() functions, both based on multi-scale fractal methods. Namely, turbulence based path function and fractional Brown motion (fBm) based path function.

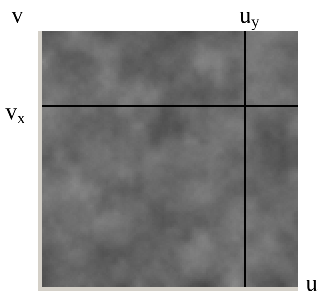

Figure 1, Illustration of Turbulence based TexturePath function

The principle of turbulence based texture growing requires one to create one 2-dimensional turbulence map in u-v Cartesian coordinates, randomly select two different straight-lines on it, and use the turbulence value along the two paths respectively as the values of x and y components of the motion vector in equation (4). This method is illustrated in Figure 1. The two black straight lines in the figure are two randomly selected paths with the method mentioned above. Then, the TexturePath() function can be defined as the following equation:

$$\begin{aligned} TexturePath_x(z_i) &= turbulence(u_i, v_x) \\ TexturePath_y(z_i) &= turbulence(u_y, v_i) \end{aligned} \quad (5)$$

Fractional Brownian motion based texture growing simulates the texture-growing path with the trace of a fractional Brownian motion. Usual Brownian motion can be easily created through integrating pseudo white noise. More general fractional Brownian motions can be simulated by multi-resolution approaches, such as random midpoint displacement method [Peitgen1988].

The principle of using random midpoint method for approximation of fBm is as follows. If the process is to be computed for times t, between 0 and 1, then one starts by setting

X(0)=0 and selecting X(1) as a sample of a Gaussian random variable with mean 0 and variance σ^2. The values of X in the interval [0, 1] can be computed recursively with the following equation:

$$X\left(\left(\frac{t_1}{2^n}+\frac{t_2}{2^n}\right)/2\right)=0.5\left[X\left(\frac{t_1}{2^n}\right)+X\left(\frac{t_2}{2^n}\right)\right]+D_n \qquad (6)$$

where Dn is a Gaussian random offset with mean 0 and variance Δ_n^2

Under the assumption

$$Var(X(t_2)-X(t_1))=\left|t_2-t_1\right|^{2H}\sigma^2 \qquad (7)$$

Δ_n^2 can be calculated with the following equation:

$$\Delta_n^2=\frac{\sigma^2}{(2^n)^{2H}}(1-2^{2H-2}) \qquad (8)$$

where H is a parameter that controls the motion property. This method can simulate general $1/f^{\beta}$ noise conveniently by choosing different H values in equation (8) [Peitgen1988].

Both turbulence based method and fBm based method prove to be able to generate reasonable results. Figure 2 illustrates two texture paths defined on the interval [0,1], one generated by turbulence based function, the other by fBm based function. The domain and the range of both functions can be scaled as needed.

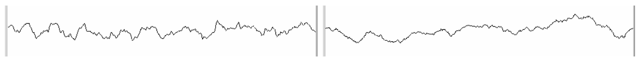

Figure 2, Turbulence based Texture path and fBm based Texture path (H=0.75)

Compared with turbulence based method, fBm based method is more powerful and flexible to give different looking paths by selecting different values of the parameter H in equation (8). Therefore, fBm based method is more attractive. Another virtue of fBm based function is that, with random midpoint displacement method, the positions of the start and end points can be assigned freely as needed. This makes it possible to tile texture cube in the z-direction, which will be discussed in detail in section 3.3.

3.3 3D texture tiling

Generally, we take a square texture image as the input texture pattern and create a 3D texture cube with the same depth as the size of the image. The method can be used for image of any sizes. Nonetheless, larger size means more computations and higher storage costs, which is expensive for many low-end machines. Here we apply tiling technique to solve this problem. That is, only a relatively small tileable cube is generated and stored, and it is tiled to form a bigger 3D texture space for final use. To achieve this goal, we begin with a tileable texture pattern. In each frame, we will give stronger turbulence to the center region of the cube, and weaken the effect of the turbulence little by little towards the frame border in a proper brim range. All edge pixels keep unchanged. This can be easily achieved by modulating the mscale parameter of equation (3) with the well-known s_curve function [Ebert1998].

$$S(t)=3t^2-2t^3 \qquad (9)$$

With such a modification, for each texture frame just generated, the pixels on x and y edges remain unchanged, and the pixels near the edge receive only constrained turbulences. Since the original image is tileable, the final cube is tileable in both x and the y directions.

We achieve tileable property in the z-direction by using an fBm based texture path generation with the positions of start point and end point assigned equally. In such a texture growing path, as the frame moves closer to the end of the cube in the z-direction, the distance from the original position in x-y plane also becomes weaker towards zero. The texture path function is modulated with equation (9) exactly as we did in x and y direction, so that the last z frame is smoothly tileable with the first z frame. Consequently, the final cube is also tileable in the z-direction.

Once the 3D texture cube is prepared, it can be tiled conveniently to any size as needed for the decoration of different 3D objects. Tiling can significantly save both computation and storage costs of solid texturing.

To summarize, our 3D texture generation algorithm can be regarded as a stochastic extrusion approach where noise technique based procedural method is used to control the growing process of the texture, not the texture itself. This makes our method distinct from other approaches, and correspondingly, a lot of interesting properties come to light.

4 Results

Our experiments are conducted via a C++ implementation running on one PC workstation equipped with a 1.6G Pentium4 processor, 256MB of RAM and an NVIDIA GeForce4 display adaptor.

Since the texture synthesis addressed in Section 3 is a solid texture technique, it is particularly suitable for decorating complex point rendering system. We borrow the QSplat system created by Rusinkiewicz et al. to test the performance of our 3D solid texturing on point rendering systems [Rusinkiewicz2000]. As in previous research [Pfister2000], we separate texture generating from rendering. The 3D texture cube is produced in a preprocessing routine, and texture mapping during rendering takes negligible time. Accordingly, the good performance of interactive rendering of QSplat is kept unchanged. Examples of 2D texture patterns used in our experiments are shown in Figure 3. Figure 4 shows the results of applying the technique to complex models. In each case, a 128*128*128 3D texture cube is synthesized from a given 128*128 texture pattern. Then several such 3D cubes are tiled to construct a larger 3D texture space of variable size. All models in Figure 4 can be interactively browsed in a 512*512 window with a frame rate of about 8 frames/second. See Table 1 for more details.

The uses of texture growing and texture turbulence are optional. Generally speaking, one or both of them can be disabled to accommodate textures with certain regular structure or to make some special effects. The wood texture of figure 3 is an example of such texture patterns. When enabled, the parameters mscale, tfactor and H offer flexible control on overall appearance.

Table 1 illustrates that the algorithm can be executed very fast. When both texture growing and texture turbulence options are enabled, the 128*128*128 3D texture cube can be constructed in around 10 seconds using our non-optimized program. Once the cube has been produced, it can be tiled to any size and decorate the point rendering system in an object-independent manner. Tiling leads to a significant saving of storage compared with conventional solid texturing techniques, and object-independent texturing results in substantial saving of computation when rendering different objects. Although the final texture space is built by tiling several copies that are exactly the same, the repetition is successfully masked by the complexity of the scene and become imperceptible. Therefore, the approach is very efficient in terms of time and space occupancy.

From Figure 4 we can see that our method is particularly suited for natural growable texture materials such as marbles and woods because the generation of the texture cube successfully emulates the growing nature of these materials. For instance, a marble vein in the vase model is successfully simulated by texture growing and texture turbulence. So the method can be used to produce vivid artistic effects over complex object to give the effects of stone or wood sculptures.

The limitation of the approach is that the seeds for our 3D texture synthesis scheme are constrained to natural growable textures. The method is not well suited for surface decorating textures because randomly looking stripes will appear in the z-direction.

5 Conclusion

We have presented a method of creating vivid 3D texture based on the principles of texture growing and texture turbulence. The algorithm is efficient and is able to generate impressive 3D texture independent of the target objects and is particularly suited for decorating of complex range scanned objects.

Acknowledgement

The work described in this paper was supported by a grant from CityU (Project # 9010005) and RGC grant No. CityU 1150/01E.

References

[Benson2002] Benson, D., and Davis, J. 2002. Octree textures. In Proceedings of ACM SIGGRAPH 2002, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 101–106.

[Cabral1993] Cabral,B., and Leedom, L. C. 1993. Imaging vector fields using line integral convolution, in Proceedings of ACM SIGGRAPH93, Computer Graphics Proceedings, Annual Conference Series, pp. 263–272.

[DeBry2002] D. DeBry, J. Gibbs, D. Petty, and N. Robins. Painting and rendering textures on unparameterized models, Proceedings of SIGGRAPH 02, 2002, 763-768.

[Ebert1998] Ebert, D., Musgrave,F.,Peachey, D., Perlin, K., AND Worley, S., Eds. 1998. Texturing and Modelling: A procedural approach. Morgan Kaufmann Publishers, 1998.

[Grossman1998] J. P. Grossman and W. Dally. Point Sample Rendering. In Rendering Techniques 98, pages 181–192. Springer, Wien, Vienna, Austria, July 1998.

[Pfister2000] Pfister, H., Zwicker, M., Vanbaar, J., Gross, M. 2000. Surfels: Surface elements as rendering primitives. In SIGGRAPH 2000, pages 335–342. New Orleans, LA, July 23-28, 2000.

[Lewis1989] Lewis, J. P., Algorithms for Solid Noise Synthesis, Computer Graphics, Vol. 23, No. 3 (SIGGRAPH89), pp. 263–270, (July 1989).

[Peachey1985] Peachey, Darwyn R., Solid Texturing of Complex Surfaces, Computer Graphics, Vol. 19, No. 3, (SIGGRAPH 85), July 1985, pp. 279–286.

[Peitgen1988] Heinz-Otto Peitgen, Dietmar Saupe, editors. The Science of Fractal Images. Springer. 1988.

[Perlin1985] Perlin, Ken, An Image Synthesizer, Computer Graphics, Vol. 19, No. 3, (SIGGRAPH 85), July 1985, pp. 287–296.

[Rusinkiewicz2000] S. Rusinkiewicz and M. Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes. In Computer Graphics, SIGGRAPH 2000 Proceedings.

[Soatto2001] Soatto, S., Doretto, G., and Wu, Y. 2001. Dynamic textures. In Proceedings of IEEE International Conference on Computer Vision, vol. 2, 439–446.

[Wang2002] Y. Z. Wang, Song Chun Zhu, A Generative Method for Textured Motion: Analysis and Synthesis, Proc. of European Conf. on Computer Vision (ECCV), Copenhagen, June 2002, pages 583-598.

[Wijk1991] Van Wijk, J. 1991. Spot noise: Texture synthesis for data visualization, Computer Graphics 25, 309–318. Proceedings ACM SIGGRAPH 91.

[Wijk2002] Jarke J. van Wijk, Image Based Flow Visualization, SIGGRAPH 2002.

Table 1, model and texture pattern details

Model	Total points / rendered points	Texture pattern (scaling size)	Texture growing	Texture turbulence	Time(s)	Main viewing direction
Vase	68097/38519	Marble1(4*4*4)	Enabled	Enabled	11.73	x
Venus	134345/62292	Marble2(4*4*4)	Enabled	Enabled	10.42	y
Horse	48485/27094	Wood(4*4*4)	Enabled	Disabled	2.98	x-z
Dragon	1279481/625726	Marble3(4*4*4)	Enabled	Enabled	10.92	x-z
Hip	530168/238934	Marble4(4*4*4)	Enabled	Enabled	10.43	x-y-z

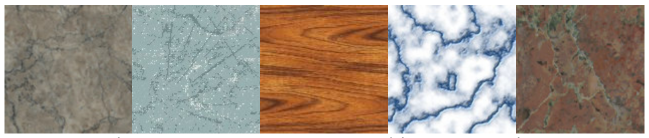
Figure 3, texture patterns used in our experiments.
From left to right: marble1, marble2, wood, marble3, marble4

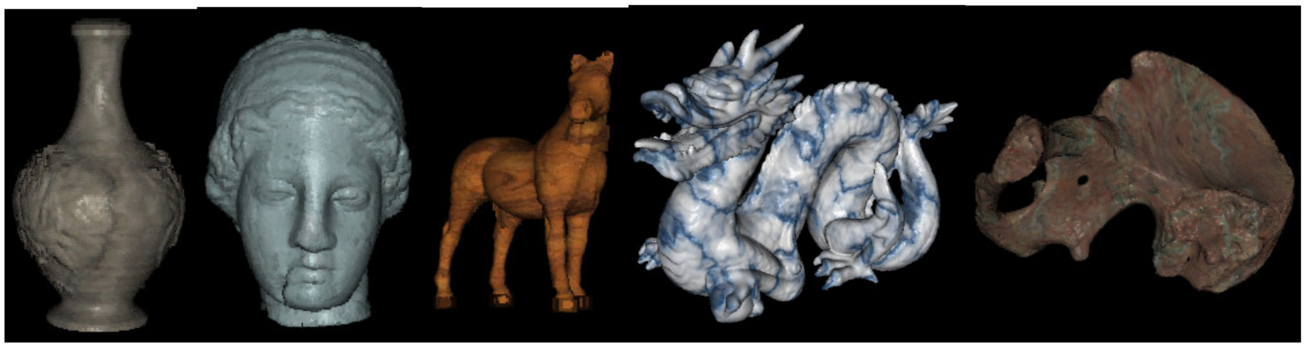
Figure 4, Results of texturing point rendering based models